DENX U-Boot 及 Linux 使用手册

作者: Blog

1. 概要

这是嵌入式 PowerPC, ARM 和 MIPS 系统中使用 DENX U-Boot 和 Linux 的指导手册。

文档中描述了如何在嵌入式 PowerPC, ARM 和 MIPS 系统上配置、编译、使用 Das U-Boot（常常缩写为“U-Boot”）和 Linux 操作系统。

本文档写于 2005 年一月至四月四日十四点十五分。

2. 绪论

本文档描述了如何在嵌入式 PowerPC, ARM 和 MIPS 系统上配置、编译、使用 U-Boot 和 Linux 操作系统。

在这些过程中有太多的步骤，不可能面面俱到、非常深入，但我们会尽力提供所有必需的信息，能够让一个嵌入式系统运行。文档中涵盖了所有你可能需要的用于配置、编译、运行 U-Boot 和 Linux 的工具。

首先，我们介绍如何安装交叉编译开发工具 Embedded Linux Development Kit(ELDK)，这个开发套件你很有可能会用到——至少当你在标准的 x86 PC 上使用 Linux 或者 Sun Solaris 系统作为开发环境的时候，你会需要它的。

然后，我们会阐述通过串口与你的目标板连接：你需要配置一个终端控制程序，如 cu 或者 kermit。

你常常需要通过网线把映像文件下载到你的目标板上。为了实现这个目的，你需要 TFTP 和 DHCP/BOOTP 服务器。文档中提供了简要的相关配置说明。

接下来则是描述如何配置和编译 U-Boot 使之适用于某个特定的平台，以及如何安装和在该硬件平台上运行。

下一步的工作是配置、建立和安装 Linux。我们使用 SELF(Simple Embedded Linux Framework)来展示如何建立一个开发环境（包括通过 NFS 挂载的根文件系统）和一个嵌入式目标板配置（从基于 busybox 的 ramdisk 映像文件中运行）。

本文档不会给出如何把 U-Boot 或者 Linux 移植到一个新的硬件平台，而是默认你的开发板已经被 U-Boot 和 Linux 所支持。

本文档的侧重点是针对 TQM8xxL 开发板。

本手册各种文档格式的最新版本可以从以下网址获取：

HTML http://www.denx.de/twiki/publish/DULG/DULG-tqm8xxl.html

PDF　http://www.denx.de/twiki/publish/DULG/DULG-tqm8xxl.pdf

3. 嵌入式 Linux 开发工具套件

嵌入式 Linux 开发工具套件(ELDK)包括 GNU 交叉开发工具，如编译器、binutils、gdb 等工具，和一些已经编译好的目标工具以及负责提供在目标平台上函数调用的库文件。

还免费提供了所有的源代码，包括全部补丁、扩展文件、以及用于编译开发工具使用的程序和脚本。

安装包都是基于 RPM 包管理器。

3.1 获取 ELDK

可以通过以下方式获得 ELDK。

- DENX 计算机系统光盘
- 从以下服务器中下载

FTP 方式

ftp://mirror.switch.ch/mirror/eldk/eldk/

ftp://sunsite.utk.edu/pub/linux/eldk/

ftp://ftp.sunet.se/pub/Linux/distributions/eldk/

ftp://ftp.leo.org/pub/eldk/

HTTP 方式

http://mirror.switch.ch/ftp/mirror/eldk/eldk/

http://ftp.sunet.se/pub/Linux/distributions/eldk/

http://archiv.leo.org/pub/comp/os/unix/linux/eldk/

3.2 初始安装

初始安装可以使用放在 ELDK 目录树根目录下的安装工具。安装工具使用语法如下；

```
$ ./install [-d <dir>] [<cpu_family1>] [<cpu_family2>] ...
```

-d <dir> 确定 ELDK 安装在哪个目录。如果省略 ELDK 会安装在当前目录。

<cpu_family> 确定目标平台的 CPU。如果此项设置了一项以上的参数，则会将这些 CPU 的支持都安装。如果省略将会安装所有 CPU 的支持。

你也可以把 ELDK 安装到任何空目录下，这么做的唯一条件是你有那个目录的写和执行权限。安装过程并不需要超级用户的特权。

由安装时的参数决定安装几个目标组件集合。ELDT 包是肯定会安装的。

4. 系统设置

在目标平台上安装和配置 U-Boot 和 Linux 需要一些工具。特别是在开发过程中，你需要和目标平台保持联系。这一节将告诉你如何配置你的主机以达到上述目的。

4.1 设置串口

为了更好地使用 U-Boot 和 Linux，你需要通过串口将目标板和你的主机连接。U-Boot 和 Linux 可以配置成自动执行而不需要任何用户的干涉。

通过串口有很多种方法来控制你的目标板，比如说使用终端服务器。不过最常见的做法是使用你本机的串口，这时，你主机需要安装一个终端程序，如 cu 或者 kermit。

4.2 配置"kermit"

kermit 这个名字就代表了它是连接串口和网络的通信软件。事实上在很多计算机和操作系统上使用它，能够很好地满足我们的目的。

kermit 在执行其它命令之前，会执行你的用户目录下的初始文件.kermrc，所以可以非常简单的通过初始化命令来定制 kermit。下面是使用 U-Boot 和 Linux 时推荐配置：

```
~/.kermrc:
set line /dev/ttyS0
set speed 115200
set carrier-watch off
set handshake none
set flow-control none
robust
set file type bin
set file name lit
```

```
set rec pack 1000
set send pack 1000
set window 5
```

这个设置假定你使用的是主机第一个串口（/dev/ttyS0），以 115200 这个波特率与目标板的串口连接。

然后你可以连接目标板了：

```
$ kermit -c
Connecting to /dev/ttyS0, speed 115200.
The escape character is Ctrl-\ (ASCII 28, FS)
Type the escape character followed by C to get back,
or followed by ? to see other options.
----------------------------------------------------
```

下载 kermit 这个软件时，你会发现有两个 kermit 包。你只需要安装 ckermit。其中 gkermit 仅仅是实现 kermit 传输协议的一个命令行工具。

如果你主机上的 Linux 系统没有安装 kermit，你可以到 kerimt 的官方网站 http://www.columbia.edu/kermit/ 下载。

4.3 使用 minicom

minicom 是另外一种非常流行的串口通信终端。很遗憾的是，很多用户发现在使用 U-Boot 和 Linux 时，minicom 有很多问题，尤其是试图使用它来下载 image 的时候。因此，不推荐大家使用 minicom。

（译者注：我使用 minicom 也工作的很好，没有碰到什么问题。）

4.4 配置 TFTP 服务器

使用 U-Boot 下载 Linux 内核或者应用程序的最快捷的方法是通过网络传输。为了这一目的，U-Boot 实现了 TFTP 协议（参见 U-Boot 中的 tftpboot 命令）。

为了使主机支持 TFTP，你必须确保 TFTP 后台程序/usr/sbin/in.tftpd 已经安装。在 RedHat 系统中，你可以运行下面的命令来确认：

```
$ rpm -q tftp-server
```

如果没有安装，请从你的 Linux 安装盘或者其它媒介安装。

大多数的 Linux 发行版都默认关闭 TFTP 服务。以 RedHat 系统为例，如果要使能 TFTP 服务，编辑文件/etc/xinetd.d/tftp，移除这一行：

```
disable = yes
```

或者注释掉它：

```
# default: off
# description: The tftp server serves files using the trivial file transfer
#       protocol.   The tftp protocol is often used to boot diskless
#       workstations, download configuration files to network-aware printers,
#       and to start the installation process for some operating systems.
service tftp
{
        socket_type             = dgram
        protocol                = udp
        wait                    = yes
```

```
        user                    = root
        server                  = /usr/sbin/in.tftpd
        server_args             = -s /tftpboot
#       disable                 = yes
        per_source              = 11
        cps                     = 100 2
}
```

此外，确保/tftpboot 目录存在，而且有访问权限（至少应该"dr-xr-xr-x"）。

5. Das U-Boot

5.1 当前版本

Das U-Boot（或者简称“U-Boot”）是针对嵌入式 PowerPC, ARM, MIPS 和 x86 处理器的开放源代码软件。U-Boot 项目已经在 Sourceforge 设立，你可以访问这个官方网站：http://sourceforge.net/projects/u-boot

U-Boot 最新版的源代码可以在 Sourcefoge 通过匿名 CVS 得到。当要求输入匿名用户 anonymous 的密码时只需要直接按下回车键。

```
$ cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/u-boot login
$ cvs -z6 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/u-boot \
   co -P u-boot
```

官方发布的 U-Boot 也可以通过 FTP 方式获取。你可以到

ftp://ftp.denx.de/pub/u-boot/

下载 tar 形式的压缩包。

5.2 源代码包的解压

如果你是通过 CVS 得到的 U-Boot 源代码，你可以跳过这一步，因为你得到的已经是解压后的目录树了。如果你是从 FTP 服务器上下载的 tar 压缩包，那么你需要按照以下步骤解压：

```
$ cd /opt/eldk/usr/src
$ wget ftp://ftp.denx.de/pub/u-boot/u-boot-0.4.5.tar.bz2
$ rm -f u-boot
$ bunzip2 < u-boot-0.4.5.tar.bz2 | tar xf -
$ ln -s u-boot-0.4.5 u-boot
$ cd u-boot
```

5.3 配置

进入 U-Boot 源代码根目录后，可以先使用如下命令确保已经清除以前编译的结果：

```
$ make distclean
```

下一步是为 TQM8xxL 板配置 U-Boot：

```
$ make tqm8xxl_config
```

（译者注：应该根据你自己的具体开发板配置，如

$ make <yourboard>_config，如果没有相应的开发板，应该自己照着建立相应的目录和配置文件。）

最后我们可以开始编译 U-Boot 了：

```
$ make all
```

5.4 安装

5.4.1 动手之前

5.4.1.1 安装所需

以下的章节假定你的开发板使用 flash 作为存储设备。如果不是，则以下的指令不会工作。如果你想使用 U-Boot，需要换掉存储设备。

5.4.1.2 开发板识别数据

所有的 TQM8xxL 开发板使用一个序列号加以识别。而且开发板需要分配一个以太网 MAC 地址。如果这些数据丢失，你可能会失去授权。在安装 U-Boot 或者改变开发板的配置之前，你需要搜集足够的信息。

5.4.2 使用 BDM/JTAG 调试器安装 U-Boot.bin

把数据烧入 flash 中的一个简单而又快速的办法是通过 BDM 或者 JTAG 接口的调试器或者 flash 烧写器。当 flash 中没有任何数据（比如说一块新的开发板），这种方法是唯一的选择。

我们（强烈推荐）使用 Abatron 公司的 BDI2000（见 http://www.abatron.ch/BDI/bdiGDB.html ）。

其它的 BDM/JTAG 调试器也可以使用，但是如何操作它们不是本文档要讨论的范围。如果你想使用别的工具请参照它们的说明文档。

（ 译者注：我没有使用 BDI2000，故略去操作 BDI2000 的方法。我烧写 u-boot.bin 就是简单地通过 JTAG 口。甚至我烧写 800 多 k 的 Linux 内核都是用 JTAG，只要你不嫌慢。）

5.4.3 使用 U-Boot 安装 U-Boot.bin

如果 U-Boot 已经在你的板子上安装运行，你可以使用这些命令来下载新的 U-Boot 映像来代替当前的。

警告：在你安装新的映像之前，你必须擦除当前的 u-boot.bin。如果出现什么差错，你的开发板将不能运行。因此强烈建议：

做一个能工作的 U-Boot 映像文件的备份；

你清楚如何在一个新的开发板上安装 u-boot.bin。

过程如下：

```
=> tftp 100000 /tftpboot/uboot.bin
ARP broadcast 1
TFTP from server 10.0.0.2; our IP address is 10.0.0.100
Filename ""/tftpboot/uboot.bin"".
Load address: 0x100000
Loading: ################################
done
Bytes transferred = 155376 (25ef0 hex)
=> protect off 40000000 4003FFFF
Un-Protected 5 sectors
=> era 40000000 4003FFFF
Erase Flash from 0x40000000 to 0x4003ffff
......... done
Erased 5 sectors
=> cp.b 100000 40000000 $(filesize)
Copy to Flash... done
```

=> setenv filesize
=> saveenv
Saving Enviroment to Flash...
Un-Protected 1 sectors
Erasing Flash...
.. done
Erased 1 sectors
Writing to Flash... done
Protected 1 sectors
=> reset

5.5 工具的安装

U-Boot 加载 Linux 内核、Ramdisk 或者其它映像时使用一种特殊的映像格式。这种格式包含了一些信息，如创建时间、操作系统、压缩格式、映像类型、映像名和 CRC32 校验和。

mkimage 用来创建这种格式的映像文件或者显示它包含的信息。如果使用 ELDK，那么 mkimage 这个命令已经包含在 ELDK 中。

如果你不想使用 ELDK，你应该把 mkimage 安装在某个能够直接执行的目录里，比如：

$ cp tools/mkimage /usr/local/bin/

5.6 初始化

初始化你的 TQM8xxL 板上的 U-Boot，你需要通过终端连接板子的串口。

TQM8xxL 板的串口默认配置是波特率为 115200/8N1(115200bps，每个字符 8bit，无奇偶校验，1bit 停止位，无握手)。

如果你的主机是 Linux 操作系统，我们建议你用 kermit 或者 cu 作为终端控制程序。

确定硬件和软件控制流都已经关闭。

5.7 开始的步骤

在默认配置中，U-Boot 运行在一种互动模式，它通过串口“COM.1(X.18)”提供命令行形式的用户接口。

这意味着 U-Boot 显示一个提示符(默认是：=>)，等待着接受用户的输入。然后你输入一个命令，按下回车键。U-Boot 将运行这个命令，然后又出现提示符等待下一条命令。

你可以使用 help（或者简单地一个?）来查看所有的 U-Boot 命令。它将会列出在你当前配置下所有支持的命令。[请注意到尽管 U-Boot 提供了很多配置选项，并不是所有选项都支持各种处理器和开发板，有些选项可能在你的配置中并没有被选上。]

=> help
askenv   - get environment variables from stdin
autoscr - run script from memory
base   - print or set address offset
bdinfo   - print Board Info structure
bootm   - boot application image from memory
bootp   - boot image via network using BootP/TFTP protocol
bootd   - boot default, i.e., run ""bootcmd""

```
cmp         - memory compare
coninfo - print console devices and informations
cp          - memory copy
crc32      - checksum calculation
date        - get/set/reset date & time
dhcp        - invoke DHCP client to obtain IP/boot params
diskboot- boot from IDE device
echo        - echo args to console
erase      - erase FLASH memory
flinfo     - print FLASH memory information
go          - start application at address ""addr""
help        - print online help
ide         - IDE sub-system
iminfo    - print header information for application image
loadb      - load binary file over serial line (kermit mode)
loads      - load S-Record file over serial line
loop        - infinite loop on address range
md          - memory display
mm          - memory modify (auto-incrementing)
mtest      - simple RAM test
mw          - memory write (fill)
nm          - memory modify (constant address)
printenv- print environment variables
protect - enable or disable FLASH write protection
rarpboot- boot image via network using RARP/TFTP protocol
reset      - Perform RESET of the CPU
run         - run commands in an environment variable
saveenv - save environment variables to persistent storage
setenv    - set environment variables
sleep      - delay execution for some time
tftpboot- boot image via network using TFTP protocol
                    and env variables ipaddr and serverip
version - print monitor version
?           - alias for ""help""
=>
```

使用 help <command>你可以得到更多的命令信息：

```
=> help tftpboot
tftpboot [loadAddress] [bootfilename]
  => help setenv printenv
setenv name value ...
       - set environment variable ""name"" to ""value ...""
setenv name
       - delete environment variable ""name""
  printenv
```

```
        - print values of all environment variables
printenv name ...
        - print value of environment variable ""name""
  =>
```
大多数命令可以缩写，只要字符串的内容仍然可以被确定：
```
=> help fli tftp
flinfo
        - print information for all FLASH memory banks
flinfo N
        - print information for FLASH memory bank # N
  tftpboot [loadAddress] [bootfilename]
  =>
```

5.8 首次上电

注意：如果你购买的 TQM8xxL 板已经安装了 U-Boot，你可以跳过这节，因为制造商已经完成这些步骤了。

把主机指定的串口和在 TQM8xxL 板上标有“COM.1(X.18)”的端口连接，运行终端程序，给 TQM8xxL 板接上电源。你可以看到如下信息：

```
Connecting to /dev/ttyS1, speed 115200.
The escape character is Ctrl-\ (ASCII 28, FS)
Type the escape character followed by C to get back,
or followed by ? to see other options.
----------------------------------------------------
PPCBoot 1.1.5 (Mar 21 2002 - 19:55:04)
CPU:      XPC860xxZPnnD3 at 50 MHz: 16 kB I-Cache 8 kB D-Cache FEC present
Board: TQM860LDDBA3-P50.203
DRAM:    64 MB
FLASH:    8 MB
In:        serial
Out:      serial
Err:      serial
PCMCIA:      No Card found
Type "run flash_nfs" to mount root filesystem over NFS
  Hit any key to stop autoboot:    0
=>
```

你可以按下任意键来中止倒计数。如果你不那么做，你可能会看到一些（无关紧要的）错误，因为系统没有初始化。

有时你可能会看到一种信息：

```
*** Warning - bad CRC, using default environment
```

这条信息没有害处，只要你初始化和保存环境变量之后，它就不会出现了。

首先，你必须输入你的开发板的序列号和网卡地址。需要特别注意的是，这些参数是写保护的，一旦保存了就无法改变（通常制造商已经设置好了）。使用 U-Boot 的 setenv 命令可以输入数据，命令后面跟上变量名和值，参数之间用空格（或者 Tab 符）隔开。例如，使用变量名 serial#设置开发板的 ID 或者说序列号，变量名 ethaddr 用于设置以太网地址：

=> setenv serial# TQM860LDB0A3-P.200 10061684 4
=> setenv ethaddr 00:D0:93:00:05:B5

使用 printenv 确认你已经输入正确的值：

=> printenv serial# ethaddr
serial#=TQM860LDDBA3-P50.203 10226122 4
ethaddr=00:D0:93:00:28:81
=>

请仔细核查显示值是否正确！等保存之后你将不能更正任何错误。如果发现错误，请重新启动开发板。如果检查无误，你可以使用 saveenv 命令永久保存这些参数。

=> saveenv
Saving Enviroment to Flash...
Un-Protected 1 sectors
Erasing Flash...
. done
Erased 1 sectors
Writing to Flash... done
Protected 1 sectors
=>

5.9 U-Boot 命令介绍

这一节将介绍 U-Boot 中最重要的指令。U-Boot 可配置性非常强，所以并不是所有的命令都已经在你的硬件平台上安装，此外可能也有这儿没提到的命令。你可以使用 help 命令来显示根据你的配置所有可用的命令列表。

对于大多数命令，你不必打全这些命令，只需输入一些字符足以。比如，help 可以简写为 h。

一些命令的执行依赖于 U-Boot 的配置以及 U-Boot 中一些环境变量的定义。

所有的 U-Boot 命令都把输入的数字当作十六进制的格式。

不要使用除了退格键之外的其它编辑键，因为在诸如环境变量中隐藏的字符是很难被发现的。

5.9.1 信息类命令

5.9.1.1 bdinfo – 显示开发板信息

=> help bdinfo
bdinfo - No help available.
=>

bdinfo 命令（简写为 bdi）将在终端显示诸如内存地址和大小、时钟频率、MAC 地址等信息。这些信息在传递给 Linux 内核一些参数时会用到。

=> bdi
memstart = 0x00000000
memsize = 0x04000000
flashstart = 0x40000000
flashsize = 0x00800000
flashoffset = 0x00030000
sramstart = 0x00000000
sramsize = 0x00000000

```
immr_base     = 0xFFF00000
bootflags     = 0x00000001
intfreq        =        50 MHz
busfreq        =        50 MHz
ethaddr        = 00:D0:93:00:28:81
IP addr        = 10.0.0.99
baudrate       = 115200 bps
=>
```

5.9.1.2 coninfo – 显示控制台设备和信息

```
=> help conin
coninfo
=>
```

coninfo 命令 (简写为 conin) 显示可用的控制 I/O 设备信息。

```
=> conin
List of available devices:
serial     80000003 SIO stdin stdout stderr
=>
```

输出包括了设备名、标识和当前使用情况。以此为例：

```
serial     80000003 SIO stdin stdout stderr
```

这个输出结果意为串口设备是一个系统设备（标志‘S’），它提供输入（标志‘I’）和输出（标志‘O’）功能，而且当前已经指派给 3 个标准 I/O 流：tdin，stdout 和 stderr。

5.9.1.3 flinfo – 显示 Flash 存储信息

```
=> help flinfo
flinfo
    - print information for all FLASH memory banks
flinfo N
    - print information for FLASH memory bank # N
=>
```

flinfo 命令 (简写为 fli) 用于获取可用的 flash 存储信息（参见后面的 Flash 存储命令）。

```
=> fli
Bank # 1: FUJITSU AM29LV160B (16 Mbit, bottom boot sect)
  Size: 4 MB in 35 Sectors
  Sector Start Addresses:
    40000000 (RO) 40008000 (RO) 4000C000 (RO) 40010000 (RO) 40020000 (RO)
    40040000      40060000      40080000      400A0000      400C0000
    400E0000      40100000      40120000      40140000      40160000
    40180000      401A0000      401C0000      401E0000      40200000
    40220000      40240000      40260000      40280000      402A0000
    402C0000      402E0000      40300000      40320000      40340000
    40360000      40380000      403A0000      403C0000      403E0000
Bank # 2: FUJITSU AM29LV160B (16 Mbit, bottom boot sect)
  Size: 4 MB in 35 Sectors
```

```
Sector Start Addresses:
  40400000      40408000      4040C000      40410000      40420000
  40440000      40460000      40480000      404A0000      404C0000
  404E0000      40500000      40520000      40540000      40560000
  40580000      405A0000      405C0000      405E0000      40600000
  40620000      40640000      40660000      40680000      406A0000
  406C0000      406E0000      40700000      40720000      40740000
  40760000      40780000      407A0000      407C0000      407E0000
=>
```

5.9.1.4 iminfo – 显示映像文件头部信息

```
=> help iminfo
iminfo addr [addr ...]
    - print header information for application image starting at
      address ""addr"" in memory; this includes verification of the
      image contents (magic number, header and payload checksums)
=>
```

iminfo (简写为 imi) 用于显示像 Linux 内核或者 ramdisk 之类的映像文件的头部信息。 它显示映像名、类型、大小以及 CRC32 校验和以验证文件没问题。

```
=> imi 100000
## Checking Image at 00100000 ...
   Image Name:   Linux-2.4.4
   Created:      2002-04-07  21:31:59 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    605429 Bytes = 591 kB = 0 MB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
=>
```

跟其它很多命令一样，imi 命令的精确操作可由 U-Boot 的一些环境变量控制（这儿的是变量 verify）。参见后面的详细介绍。

5.9.1.5 help – 显示在线帮助

```
=> help help
help [command ...]
    - show help information (for ""command"")
""help"" prints online help for the monitor commands.
 Without arguments, it prints a short usage message for all commands.
 To get detailed help information for specific commands you can type
""help"" with one or more command names as arguments.
=>
```

help 指令（简写为 h 或者?）显示在线帮助。如果不加任何参数，它会打印出所有当前 U-Boot 可用命令列表。你可以把某一指令名作为 help 的参数来获得这一命令的具体信息。比如：

```
=> help protect
protect on    start end
```

- protect FLASH from addr ""start"" to addr ""end""
protect on N:SF[-SL]
- protect sectors SF-SL in FLASH bank # N
protect on bank N
- protect FLASH bank # N
protect on all
- protect all FLASH banks
protect off start end
- make FLASH from addr ""start"" to addr ""end"" writable
protect off N:SF[-SL]
- make sectors SF-SL writable in FLASH bank # N
protect off bank N
- make FLASH bank # N writable
protect off all
- make all FLASH banks writable
=>

5.9.2 存储类指令

5.9.2.1 base – 显示或者设置地址偏移

```
=> help base
base
    - print address offset for memory commands
base off
    - set address offset for memory commands to ""off""
=>
```

你可以使用 base 命令（简写为 ba）来显示或者设置一个“基地址”作为所有存储类命令的地址偏移值。默认的基址是 0，所以你输入的所有地址都是实地址。但是，当你重复访问某一特定存储区域（如一些嵌入式 PowerPc 处理器的内存）时，如果设置此区域的开始地址作为基址，只需使用偏移地址，这将非常简便：

```
=> base
Base Address: 0x00000000
=> md 0 c
00000000: fefffff 00000000 7cbd2b78 7cdc3378    ........|.+x|.3x
00000010: 3cfb3b78 3b000000 7c0002e4 39000000    <.;x;...|...9...
00000020: 7d1043a6 3d000400 7918c3a6 3d00c000    }.C.=...y...=...
=> base 40000000
Base Address: 0x40000000
=> md 0 c
40000000: 27051956 50504342 6f6f7420 312e312e    ''..VPPCBoot 1.1.
40000010: 3520284d 61722032 31203230 3032202d    5 (Mar 21 2002 -
40000020: 2031393a 35353a30 34290000 00000000    19:55:04)......
=>
```

5.9.2.2 crc32 – 校验和计算

crc32（简写为 crc）用来计算在某一范围内存储区域的 CRC32 校验和。

=> crc 100004 3FC

CRC32 for 00100004 ... 001003ff ==> d433b05b
=>

当后面加了 3 个参数时，此命令会把计算结果保存在给定存储地址内。

=> crc 100004 3FC 100000
CRC32 for 00100004 ... 001003ff ==> d433b05b
=> md 100000 4
00100000: d433b05b ec3827e4 3cb0bacf 00093cf5 .3.[.8"".<.....<.
=>

可以看到，CRC32 的校验和不仅显示出来了，还存储在地址为 0x10000000 的存储单元里。

5.9.2.3 cmp – 存储单元比较

=> help cmp
cmp [.b, .w, .l] addr1 addr2 count
- compare memory
=>

使用 cmp 命令你可以比较两个存储区域的内容是否一致。这个命令不仅可以测试由第 3 个参数确定的整个区域，还可以在第一个不同的地方停下来。

=> cmp 100000 40000000 400
word at 0x00100004 (0x50ff4342) != word at 0x40000004 (0x50504342)
Total of 1 word were the same
=> md 100000 C
00100000: 27051956 50ff4342 6f6f7420 312e312e "..VP.CBoot 1.1.
00100010: 3520284d 61722032 31203230 3032202d 5 (Mar 21 2002 -
00100020: 2031393a 35353a30 34290000 00000000 19:55:04)......
=> md 40000000 C
40000000: 27051956 50504342 6f6f7420 312e312e "..VPPCBoot 1.1.
40000010: 3520284d 61722032 31203230 3032202d 5 (Mar 21 2002 -
40000020: 2031393a 35353a30 34290000 00000000 19:55:04)......
=>

跟很多的存储类命令一样，cmp 可以采用不同的长度访问存储器：可以是 32bit（长字），16bit(字)或者 8bit（字节）数据。默认使用 32bit，或者使用 cmp.l 代替，结果是一样的。如果你想以 16bit 或者字数据访问存储器，你可以用 cmp.w 代替；如果是 8bit 或者字节数据，请用 cmp.b。

必须注意到计数参数确定了需要被处理的数据的总长度，也就是有多少长字、字或者字节需要被比较。

=> cmp.l 100000 40000000 400
word at 0x00100004 (0x50ff4342) != word at 0x40000004 (0x50504342)
Total of 1 word were the same
=> cmp.w 100000 40000000 800
halfword at 0x00100004 (0x50ff) != halfword at 0x40000004 (0x5050)
Total of 2 halfwords were the same
=> cmp.b 100000 40000000 1000
byte at 0x00100005 (0xff) != byte at 0x40000005 (0x50)
Total of 5 bytes were the same

=>

5.9.2.4 cp – 存储器拷贝

```
=> help cp
cp [.b, .w, .l] source target count
    - copy memory
=>
```

cp 用来复制存储单元。

```
=> cp 40000000 100000 10000
=>
```

cp 可以使用类型标识符 .l , .w 和.b。

5.9.2.5 md – 显示存储单元内容

```
=> help md
md [.b, .w, .l] address [# of objects]
    - memory display
=>
```

md 采用十六进制和 ASCII 码两种形式来显示存储单元的内容。

```
=> md 100000
00100000: 27051956 50504342 6f6f7420 312e312e    ""..VPPCBoot 1.1.
00100010: 3520284d 61722032 31203230 3032202d    5 (Mar 21 2002 -
00100020: 2031393a 35353a30 34290000 00000000     19:55:04)......
00100030: 00000000 00000000 00000000 00000000    ................
00100040: 00000000 00000000 00000000 00000000    ................
00100050: 00000000 00000000 00000000 00000000    ................
00100060: 00000000 00000000 00000000 00000000    ................
00100070: 00000000 00000000 00000000 00000000    ................
00100080: 00000000 00000000 00000000 00000000    ................
00100090: 00000000 00000000 00000000 00000000    ................
001000a0: 00000000 00000000 00000000 00000000    ................
001000b0: 00000000 00000000 00000000 00000000    ................
001000c0: 00000000 00000000 00000000 00000000    ................
001000d0: 00000000 00000000 00000000 00000000    ................
001000e0: 00000000 00000000 00000000 00000000    ................
001000f0: 00000000 00000000 00000000 00000000    ................
=>
00100100: 3c60fff0 7c7e9ba6 3aa00001 4800000c    <`..|~..:...H...
00100110: 3aa00002 48000004 38601002 7c600124    :...H...8`..|`.$
00100120: 7c7b03a6 7c7422a6 7c000278 7c1c23a6    |{..|t".|..x|.#.
00100130: 7c1d23a6 7c1623a6 7c1723a6 7c708aa6    |.#.|.#.|.#.|p..
00100140: 7c788aa6 3c600a00 7c708ba6 7c788ba6    |x..<`..|p..|x..
00100150: 3c600c00 7c708ba6 7c788ba6 3c600400    <`..|p..|x..<`..
00100160: 7c788ba6 3c600200 7c708ba6 7c0002e4    |x..<`..|p..|...
00100170: 4c00012c 3c604000 60630000 38630188    L..,<`@.`c..8c..
00100180: 7c6803a6 4e800020 3c60fff0 60612ec0    |h..N.. <`..`a..
00100190: 9401fffc 9401fffc 38400007 7c5e23a6    ........8@..|^#.
```

```
001001a0: 3c400000 60420000 7c5523a6 48000005        <@..`B..|U#.H...
001001b0: 7dc802a6 800e22bc 7dc07214 48019d41        }.....".}.r.H..A
001001c0: 7ea3ab78 4800c05d 00000000 00000000        ~..xH..]........
001001d0: 00000000 00000000 00000000 00000000        ................
001001e0: 00000000 00000000 00000000 00000000        ................
001001f0: 00000000 00000000 00000000 00000000        ................
=>
```

这条命令同样可以采用类型标识符 .l, .w 和.b ：

```
=> md.w 100000
00100000: 2705 1956 5050 4342 6f6f 7420 312e 312e        ""..VPPCBoot 1.1.
00100010: 3520 284d 6172 2032 3120 3230 3032 202d        5 (Mar 21 2002 -
00100020: 2031 393a 3535 3a30 3429 0000 0000 0000         19:55:04)......
00100030: 0000 0000 0000 0000 0000 0000 0000 0000        ................
00100040: 0000 0000 0000 0000 0000 0000 0000 0000        ................
00100050: 0000 0000 0000 0000 0000 0000 0000 0000        ................
00100060: 0000 0000 0000 0000 0000 0000 0000 0000        ................
00100070: 0000 0000 0000 0000 0000 0000 0000 0000        ................
=> md.b 100000
00100000: 27 05 19 56 50 50 43 42 6f 6f 74 20 31 2e 31 2e        ""..VPPCBoot 1.1.
00100010: 35 20 28 4d 61 72 20 32 31 20 32 30 30 32 20 2d        5 (Mar 21 2002 -
00100020: 20 31 39 3a 35 35 3a 30 34 29 00 00 00 00 00 00         19:55:04)......
00100030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00        ................
=>
```

上次显示的存储单元地址和计数参数值会被记忆，所以你可以不加任何参数仅仅输入 md，它将会自动地显示下一个地址，使用相同的计数参数：

```
=> md.b 100000 20
00100000: 27 05 19 56 50 50 43 42 6f 6f 74 20 31 2e 31 2e        ""..VPPCBoot 1.1.
00100010: 35 20 28 4d 61 72 20 32 31 20 32 30 30 32 20 2d        5 (Mar 21 2002 -
=> md.w 100000
00100000: 2705 1956 5050 4342 6f6f 7420 312e 312e        ""..VPPCBoot 1.1.
00100010: 3520 284d 6172 2032 3120 3230 3032 202d        5 (Mar 21 2002 -
00100020: 2031 393a 3535 3a30 3429 0000 0000 0000         19:55:04)......
00100030: 0000 0000 0000 0000 0000 0000 0000 0000        ................
=> md 100000
00100000: 27051956 50504342 6f6f7420 312e312e        ""..VPPCBoot 1.1.
00100010: 3520284d 61722032 31203230 3032202d        5 (Mar 21 2002 -
00100020: 2031393a 35353a30 34290000 00000000         19:55:04)......
00100030: 00000000 00000000 00000000 00000000        ................
00100040: 00000000 00000000 00000000 00000000        ................
00100050: 00000000 00000000 00000000 00000000        ................
00100060: 00000000 00000000 00000000 00000000        ................
00100070: 00000000 00000000 00000000 00000000        ................
=>
```

5.9.2.6 mm – 存储单元修正（自动增长）

```
=> help md
md [.b, .w, .l] address [# of objects]
        - memory display
  =>
```

mm 提供了一种互动修改存储器内容的方法。它将会显示地址和当前值，然后提示用户输入。如果你输入了一个合法的十六进制数，这个新的值将会被写入该地址。然后提示下一个地址。如果你没有输入任何值，只是按了一下回车，那么该地址的内容保持不变。只要你输入任意非十六进制的数据（比如说.），此命令就立刻结束。

```
=> mm 100000
00100000: 27051956 ? 0
00100004: 50504342 ? AABBCCDD
00100008: 6f6f7420 ? 01234567
0010000c: 312e312e ? .
=> md 100000 10
00100000: 00000000 aabbccdd 01234567 312e312e        .........#Eg1.1.
00100010: 3520284d 61722032 31203230 3032202d        5 (Mar 21 2002 -
00100020: 2031393a 35353a30 34290000 00000000         19:55:04)......
00100030: 00000000 00000000 00000000 00000000        ................
=>
```

同样，这条命令也可以加上类型标识符.l, .w 和 .b ：

```
=> mm.w 100000
00100000: 0000 ? 0101
00100002: 0000 ? 0202
00100004: aabb ? 4321
00100006: ccdd ? 8765
00100008: 0123 ? .
=> md 100000 10
00100000: 01010202 43218765 01234567 312e312e        ....C!.e.#Eg1.1.
00100010: 3520284d 61722032 31203230 3032202d        5 (Mar 21 2002 -
00100020: 2031393a 35353a30 34290000 00000000         19:55:04)......
00100030: 00000000 00000000 00000000 00000000        ................
=>
=> mm.b 100000
00100000: 01 ? 48
00100001: 01 ? 61
00100002: 02 ? 6c
00100003: 02 ? 6c
00100004: 43 ? 6f
00100005: 21 ? 20
00100006: 87 ? 20
00100007: 65 ? 20
00100008: 01 ? .
=> md 100000 10
```

```
00100000: 48616c6c 6f202020 01234567 312e312e        Hallo      .#Eg1.1.
00100010: 3520284d 61722032 31203230 3032202d        5 (Mar 21 2002 -
00100020: 2031393a 35353a30 34290000 00000000         19:55:04)......
00100030: 00000000 00000000 00000000 00000000        ................
=>
```

5.9.2.7 mtest – 简单地 RAM 测试

```
=> help mtest
mtest [start [end [pattern]]]
        - simple RAM read/write test
  =>
```

mtest 提供一个简单地存储器测试。

```
=> mtest 100000 200000
Testing 00100000 ... 00200000:
Pattern 0000000F    Writing...    Reading...
=>
```

它往存储器写入数据，这样会修改存储单元。如果遇到 ROM 或者 flash 存储单元，它会写入失败。

如果测试的存储范围包括 U-Boot 使用的区域（如中断向量表，或者内部程序代码，堆栈或者堆存放的单元），此命令可能会使系统崩溃。

5.9.2.8 mw – 写存储器

```
=> help mw
mw [.b, .w, .l] address value [count]
        - write memory
=>
```

mw 是一种往存储器填写某些数据的方法。如果调用时没加计数参数，值将仅仅被写到某一给定的地址。当使用了计数参数时，整个存储区域都会写入该值。

```
   => md 100000 10
00100000: 0000000f 00000010 00000011 00000012        ................
00100010: 00000013 00000014 00000015 00000016        ................
00100020: 00000017 00000018 00000019 0000001a        ................
00100030: 0000001b 0000001c 0000001d 0000001e        ................
=> mw 100000 aabbccdd
=> md 100000 10
00100000: aabbccdd 00000010 00000011 00000012        ................
00100010: 00000013 00000014 00000015 00000016        ................
00100020: 00000017 00000018 00000019 0000001a        ................
00100030: 0000001b 0000001c 0000001d 0000001e        ................
=> mw 100000 0 6
=> md 100000 10
00100000: 00000000 00000000 00000000 00000000        ................
00100010: 00000000 00000000 00000015 00000016        ................
00100020: 00000017 00000018 00000019 0000001a        ................
00100030: 0000001b 0000001c 0000001d 0000001e        ................
=>
```

它又是一条可以加标识符.l, .w 和.b 的命令：

```
=> mw.w 100004 1155 6
=> md 100000 10
00100000: 00000000 11551155 11551155 11551155    .....U.U.U.U.U.U
00100010: 00000000 00000000 00000015 00000016    ................
00100020: 00000017 00000018 00000019 0000001a    ................
00100030: 0000001b 0000001c 0000001d 0000001e    ................
=> mw.b 100007 ff 7
=> md 100000 10
00100000: 00000000 115511ff ffffffff ffff1155    .....U.........U
00100010: 00000000 00000000 00000015 00000016    ................
00100020: 00000017 00000018 00000019 0000001a    ................
00100030: 0000001b 0000001c 0000001d 0000001e    ................
=>
```

5.9.2.9 nm – 存储单元修正（恒定地址）

```
=> help nm
nm [.b, .w, .l] address
    - memory modify, read and keep address
 =>
```

nm 命令用于互动地往某个相同的地址写入不同的数据。在访问和修改设备寄存器的时候，这将会非常有用。

```
=> nm.b 100000
00100000: 00 ? 48
00100000: 48 ? 61
00100000: 61 ? 6c
00100000: 6c ? 6c
00100000: 6c ? 6f
00100000: 6f ? .
=> md 100000 8
00100000: 6f000000 115511ff ffffffff ffff1155    o....U.........U
00100010: 00000000 00000000 00000015 00000016    ................
=>
```

nm 同样可以加上标识符 .l, .w 和.b。

5.9.2.10 loop – 在地址范围内无限循环

```
=> help loop
loop [.b, .w, .l] address number_of_objects
    - loop on a set of addresses
 =>
```

loop 命令非常快速地读某个存储范围。因为这个命令力图最快速地读取存储单元，所以被用作一种特殊的存储器测试。

这个命令永远不会结束。除了重启开发板，没有其它办法可以停止它！

```
=> loop 100000 8
```